The CEDPS Troubleshooting Architecture and Deployment on the Open Science Grid

Brian L. Tierney, Dan Gunter: Lawrence Berkeley National Laboratory

Jennifer M. Schopf: Argonne National laboratory

Corresponding Author: bltierney@lbl.gov

Abstract. Tracking failures and poor performance across a widely distributed system of resources has proven challenging for many ongoing DOE applications. An example is the Open Science Grid (OSG) project, which currently experiences a roughly 15% job failure rate. This can be an issue not only for Grid computing but for anyone performing large-scale data transfers to remote machines because of the large number of interconnected components and services.

As part of the Center for Enabling Distributed Petascale Science (CEDPS) project we have been building an infrastructure to work with current middleware and existing system tools to more easily track failures and discover anomalous behavior. This consists of a common logging format, the extension of syslog-ng for centralized collection of data, a data summarizer to more easily manage the volume of logging, and an anomaly detection system that can connect to a warning system when unexpected behaviors occur. We are currently working with OSG to deploy a prototype of the full system. The initial logs gathered will be used to extend the analysis tools and to increase the reliability of the services for the SciDAC end user community.

1. Introduction

Many of today's Grids have ongoing performance and reliability problems that have yet to be addressed. Grid2003, now Open Science Grid (OSG) [2], saw a 30% job submission failure rate [4], with 90% of the failures caused by problems such as disk filling errors, gatekeeper overloading, and network interruptions. This error rate has reduced in the past three years to around 15%, with the current goal of attaining a 90% success rate this year. Yet clearly having one of every ten job submissions fail is not truly acceptable in a production setting.

Troubleshooting Grid middleware can be difficult because of the large number of interconnected components. For example, a single action, such as reliably transferring a directory of files, can result in the coordination of a wide suite of loosely coupled software tools. These include security software to handle the certificates, check permissions, perform delegation, and possibly encrypt the message streams, file transfer tools to check the disk space, set up the connections, and transfer data between resources, and reliability software that must understand retry policies, track transfer status behavior, and react to failures. Each of these systems typically creates its own log files in its own log format, and combining log information from several components in order to understand what caused a given failure can be challenging. However, this is exactly what is needed to troubleshoot a problem as it cascades from one component into the next.

Sufficiently detailed log data is often not available in any form. For real-time debugging of interleaved and interrelated software stacks, we often need an execution trace, but logging at that level of detail can easily become unmanageable. For example, a full trace of the I/O operations performed by a single GridFTP server [1] capable of saturating a 10-gigabit network will generate O(20,000) log events per second, or over 70 million per hour. If Grid middleware components generally ran this level of detailed monitoring, the perturbation of the systems would be unacceptable. And yet logging only very coarse-grained information and asynchronous status messages, as is the common practice today, makes debugging failures a heroic effort.

Because of its complexity and heterogeneity, Grid middleware lags behind stand alone system tools in terms of anomaly detection, where an anomaly is defined as an unexpected degradation in behavior that adversely affects application performance. Detecting failures is made more difficult by fault tolerance mechanisms such as retries in GridFTP transfers and most workflow engines, which may mask more serious errors. Detecting performance degradation is also complicated by high system variance and periodic patterns that last days or weeks. These characteristics underscore the importance of doing problem detection in real time.

2. A Distributed Systems Troubleshooting Infrastructure

We are targeting an environment in which the monitored components, including long-running services and applications, are managed under separate administrative domains. For security, privacy, and manageability reasons, the logs of these components may be shared only partially. Networks between the monitored components and the outside world may also be slow, unreliable, or both.

An example of one such large distributed system is the Open Science Grid (OSG), a Grid infrastructure currently comprising over 75 international sites. Services can have uptimes measured in days or weeks, and application developers regularly use tens of sites at a time. Failure rates can be quite high, and debugging a system problem can take days due to inaccessible logs or lack of detailed information.

Our basic approach is to instrument middleware following a set of logging best practices, aggregate and filter logs with *syslog-ng* [11], and then simultaneously analyze the streaming data while archiving it to a relational database. We believe this provides a logging infrastructure that allows for scalable analysis of the distributed logs. Each component is described in more detail below.

2.1. Logs

We have written a recommendations guide for Logging Best Practices [7] that combines good instrumentation practices with log format guidelines.

All logs should contain a unique event name and an ISO-format timestamp [8], and all system operations that might fail or experience performance variations should be wrapped with *start* and *end* events. All logs from a given execution thread must be tagged with a globally unique ID (or GUID), such as a Universal Unique Identifiers (UUIDs) [9].

As to format, logs should be composed of lines of ASCII name=value pairs. This is highly portable, human-readable, and works well with line-oriented tools.

2.2. Basic Deployment

Troubleshooting analysis often requires detailed comparison of distributed logged information. A distributed query across administrative domains can be difficult to deploy in production Grids such as OSG. Instead, we aggregate the data using the open source tool syslog-ng, which enables us to filter logs based on program name, log level, and even a regular expression on message contents (which is particularly easy to apply to a name=value format). Loading a subset of logs into a relational database enables sophisticated data mining. Historical queries can provide baseline performance information, and this database can be used for post-mortem anomaly detection.

Streaming analysis of the data is more amenable to online anomaly detection. Syslog-ng can redirect a subset of the events (based on program name, etc.) to an analysis engine, such as a simple missing event detector. If all important actions are wrapped with start and end events as recommended, then a large class of troubleshooting problems can easily be found by simply looking for missing end events, which can indicate that something failed without generating an error, or that something is taking far too long to complete.

This infrastructure gives the basic functionality to begin to do troubleshooting on large Grids such as OSG. It also opens the door to begin work on some higher level troubleshooting tools such as a more complex analysis engine that does anomaly detection. An example of this full system being used for a GridFTP third-party transfer is shown in Figure 1. Logs are summarized and forwarded to a central location using syslog-ng, where they are analyzed for anomalies.

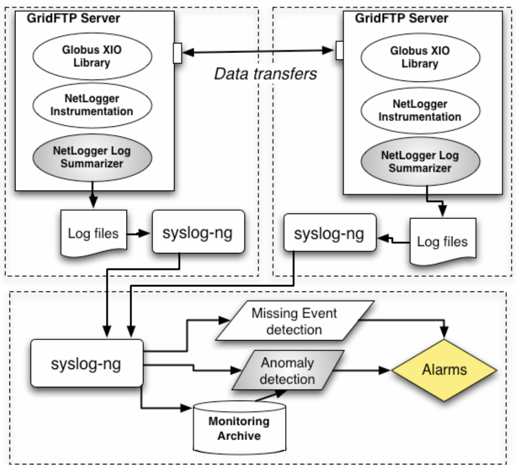

Figure 1: Log generation and collection process.

2.3. Data Summarizer

Tracing is generally avoided in production environments because it can generate too much log data and, at high frequencies, perturb the system. However, trace data is important for understanding the operation of a complex system, particularly for understanding the performance with respect to a given user. For example, detailed logging of GridFTP events as shown in Table 1 was needed to debug errors in parallel streaming performance.

Table 1: GrdFTP log events.

Sender	Receiver
disk.read.start	network.read.start
disk.read.end	network.read.end
network.write.start	disk.write,start
network.write.end	disk.write.end

2.4. Missing Event Detector

Therefore, our approach is to moderate the log creation and processing overhead of tracing, while retaining most of its benefit for fine-grained application analysis, by embedding a flexible summarization engine inside the trace library itself. We have implemented a log summarization module for NetLogger [10] that can reduce the amount of log data generated by several orders of magnitude, while still capturing key information.

The basic idea of the summarization is that many iterations of some tight loop can be compressed into a single derived event. The choice of which and how many iterations to combine is a parameter that can be chosen based on the needs of a given system. Our implementation uses a time period, so that every so many seconds, a single summary event is produced for each distinct "stream" of raw events. Variations on this theme include summarizing by number of events or waiting for a synchronous “flush” command.

Summarization also, of course, dramatically reduces log volume. For example, collecting each disk and network operation on both ends of a 1Gb/s data transfer (8 events per [256K] block if each operation is broken into a start and end event) equates to 7.2 million events per hour for full logging, compared to 3,600 events per hour for one second summary events.

2.5. Missing Event Detector

If, as described above, every interesting operation is wrapped with two log messages indicating its start and end, then a generally useful analysis is to find "start" messages that have no matching "end." We have implemented this in a component called the *missing event detector*, which we have integrated into our parsing and log analysis pipeline so that missing events can be detected in a streaming fashion and added as a new type of event in the archive. This greatly simplifies the job of online troubleshooting, as a simple relational join can show the time, location, or program-related context of all such missing events.

So far, we have come across two challenges. First, the "end" event may not appear because the developer forgot to log it. This is relatively easy to combat with decent unit tests that cover the main and failure paths throughout the code (something that is needed anyways). A more interesting problem is knowing how long to wait for the end event before declaring it missing. Often, a reasonable upper bound is known; if not, statistical methods can be used as in **Error! Reference source not found.**. We are working on both of these issues in the context of Globus logging and OSG deployment, and we are particularly conscious of the practical importance of integrating the configuration of the missing event detector with the data analysis interface(s).

2.6. MDS4 Trigger Service

One planned extension for this work is to incorporate the MDS4 Trigger service [3] to assist with more immediate responses to changes of state or missing event detection. The Trigger service collects information using WS-RF standard interfaces and compares that data against a set of conditions defined in a configuration file. When a condition is met, an action takes place, such as emailing a system administrator or updating a website.

Currently this service is deployed on the Earth System Grid project [2], where it monitors seven types of service across ten sites and reports on service failures or performance changes. This service has been helpful to increase deployment reliability by ensuring services that are down are reported automatically before they are noticed by a user. With systemwide data, a pattern of failure messages that occur close together in time can indicate a problem at a higher level. For example, failure messages indicated that hierarchical storage resource managers at three different locations failed simultaneously. Since the chance of such simultaneous failures is remote, these errors are more likely an indication of a network outage or some failure of the monitoring service or the client that queries the state of storage resource managers and hierarchical storage systems.

3. OSG Deployment

The Open Science Grid is a distributed computing infrastructure for large-scale scientific research. It is built and operated by a consortium of universities, national laboratories, scientific collaborations and software developers. The CEDPS project has been working closely with OSG to design and deploy a centralized log collection framework based on syslog-ng.

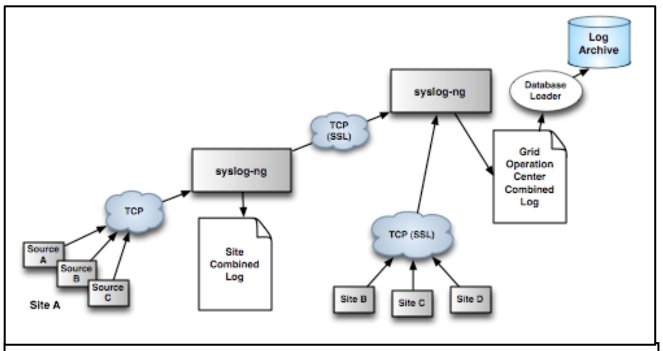

Figure 2: Sample OSG syslog-ng deployment

A particular configuration for syslog that sends logs to a central collector will be part of the next OSG software, called the Virtual Data Toolkit (VDT) [13], with a planned release in August. It is assumed that most sites will want to have their own central log repository, and that a subset of these logs will be forwarded to the OSG "Grid Operations Center" for centralized troubleshooting, auditing, and possible forensics in case of a security incident, as shown in Figure 2. Logs are then inserted into a mySQL database and viewable via a web interface.

4. Conclusion

In this paper we present an infrastructure for troubleshooting performance problems in a large distributed system such as a Grid. We introduce a new component, an application library that performs application trace summarization with minimal overhead, dramatically reducing the amount of log information while still preserving key performance information. We also discuss our work with the Open Science Grid to deploy a central log collection service for their Grid Operations Center.

Acknowledgments

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contracts DE-AC03-76SF00098 and DE-AC02-06CH11357.

References

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, “The Globus striped GridFTP framework and server,” in *Proceedings of IEEE Supercomputing,* November 2005.

[2] D. Bernholdt et al.,, “The Earth System Grid: Supporting the next generation of climate modeling research,” in *Proceedings of the IEEE*, 93 (3), 485–495, 2005.

[3] A. Chervenak, J. Schopf, et al., “Monitoring the Earth System Grid with MDS4,” in *Proceedings of the Second International Conference on eScience and Grid Computing* (eScience 2006), December 2006.

[4] I. Foster et al., “The Grid2003 production grid: Principles and practice,” *in Proceedings of the IEEE International Symposium on High Performance Distributed Computing,* 2004

[5] D. Gunter, K. Jackson, D. Konerding, J. Lee, and B. Tierney, “Essential grid workflow monitoring elements,” in *Proceedings of the International Conference on Grid Computing and Applications,* 2005.

[6] D. K. Gunter, B. L. Tierney and S. J. Bailey, “Scalable analysis of distributed workflow traces,” International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05) , LBNL-57060, 2005.

[7] “Logging best practices guide.” http://www.cedps.net/wiki/index.php/LoggingBestPractices/

[8] “Iso-8601: Data elements and interchange formats – information exchange - representation of dates and times,” International Organization for Standardization, 1988. http://www.iso.ch/markete/8601.pdf]

[9] P. Leach, M. Mealling, and R. Salz, “A universally unique identifier (uuid) urn namespace,” RFC4122, July 2005.

[10] Open Science Grid. http://www.opensciencegrid.org/

[11] syslog-ng: http://www.balabit.com/products/syslog-ng/

[12] B. Tierney and D. Gunter, “Netlogger: A toolkit for distributed system performance tuning and debugging,” LBNL, Tech. Rep. LBNL-51276, 2002.

[13] VDT: http://vdt.cs.wisc.edu/